APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: AUTOMATIC ELIMINATION OF FUNCTIONAL DEPENDENCIES BETWEEN COLUMNS

APPLICANT: JENS-PETER DITTRICH, OLAF MEINCKE, GUENTER RADESTOCK and ANDREW ROSS

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EV 398 159 256 US

February 26, 2004

Date of Deposit

AUTOMATIC ELIMINATION OF FUNCTIONAL DEPENDENCIES BETWEEN COLUMNS

BACKGROUND

[0001] The following description relates to reorganizing data in machine-readable mediums, data processing systems, and in memory to reduce the redundancy of the data and allow the available system resources to be used efficiently.

[0002] Companies handle and store large amounts of data. Such data can consume significant information technology resources, both during handling and processing, when it occupies space in memory (e.g., random access memory, RAM) and during long-term storage, when it occupies space on a machine-readable medium (e.g., hard disk or magnetic tape). For example, businesses may not only need to spend a portion of their resources on computers for workers, but for the data that the workers collect, process, and generate. Resources may also have to be spent for servers, databases, and systems to store the data, as well as warehouses and facilities to store those components. Businesses may have to allocate resources and personnel for the management of the data, and the ability to turn their data into useful organizational knowledge.

[0003] Certain techniques of organizing, formatting, or storing business data, when employed in certain business scenarios, can contribute to various amounts of data redundancy. For example, companies often store large volumes of business data in data models, such as cube-like data models, that are optimized for a large number of business scenarios. Even though the cube-like data models may offer advantages in a particular business scenario, the data models may be superfluous or irrelevant for other business scenarios. When businesses have data models for a significant amount of irrelevant scenarios, the stored data using such models may include redundancies and may occupy more space than necessary in memory. If the data can be

reorganized in such a way that the amount of redundancy is reduced, then the business can reduce the costs of storing and managing the data.

SUMMARY

[0004] In one implementation, the present disclosure relates to a computer-implemented method of reducing redundancy within a data model in a database, in which the data model is represented by at least one table. The method includes determining a number of distinct values of partial keys in a table. Each partial key represents at least one row in the table. One or more columns of the table are reordered by cardinality of partial keys, in which the cardinality of a partial key represents a number of distinct values of the partial key. The method includes determining whether pairs of partial keys are functionally dependent, and eliminating one or more columns having functional dependencies from the table.

[0005] The method may include placing one or more eliminated columns into a separate table so that the column with a highest cardinality is in the leftmost position, and the column with the lowest cardinality is in the rightmost position. The partial key K(i) can have a partial key with an index i and a value Kri for a tuple t(r) in row with index r. The number of distinct values of K(i) can include cardinality |K(i)|. The tuple t may include k key figures and d partial keys K(1), ..., K(d). A table T may have n tuples and d+k columns, in which the n tuples includes rows. A function F(x) = y may include a mapping between partial keys x and y in a same tuple, and a flag fd may have the Boolean values of true or false.

[0006] In determining whether pairs of partial keys are functionally dependent, the method may include defining function F from each partial key to every other partial key to its right in a reordered table for each row in table T. The method may also include a determination that a functional dependency exists when the function F(Kri) = Krj is the same function for each tuple

t(r) in the table for values of index i from 1 to (d – 1) and for values of j from (i +1) to d. When a tuple t is in the table T and F(Kri) is not equal to Krj, a functional dependency may not exist between columns i and j.

[0007] In determining whether pairs of partial keys are functionally dependent for each i from 1 to (d – 1) and j from (i + 1) to d, the method may include setting the flag fd to true. For each tuple t in T, the method may include determining whether F(Kri) is defined, in which F(Kri) is set equal to Krj upon determining that F(Kri) is not defined. The determining whether pairs of partial keys are functionally dependent, the method may also include looping through the tuples t in T, generating a report indicating that column i is functionally dependent on column j if flag fd is true after the looping through the tuples t in T.

[0008] In determining whether F(Kri) is defined, the method may include that upon determining that F(Kri) is defined, determining whether F(Kri) is equal to Krj, in which determining that F(Kri) is equal to Krj can permit looping through the tuples t in T. In determining that F(Kri) is not equal to Krj, the method may involve concluding that K(i) is not functionally dependent on K(j), setting flag fd to false, and breaking the looping through the tuples t in T.

[0009] In another implementation, the present disclosure relates to a computer-implemented method of reducing redundancy within a data model in a database, in which the data model is represented by at least one table. The method includes determining a number of distinct values of partial keys in a table, and reordering one or more columns of the table by cardinality of partial keys, in which each partial key represents at least one row in the table. The cardinality of a partial key represents a number of distinct values of the partial key. The method also includes determining whether pairs of partial keys are functionally dependent, eliminating one or more

columns having functional dependencies from the table, and creating an exception list for the pairs of partial keys that are not functionally dependent.

[0010] The partial key K(i) may include a partial key with an index i for a tuple t, in which the number of distinct values of K(i) includes cardinality |K(i)|. The tuple t may include k key figures and d partial keys K(i) for i from 1 to d, and a table T may include n tuples and (d + k) columns. The n tuples may include rows, and a function F(x) = y may have a mapping between partial keys x and y in a same tuple. The exception list for the pairs of partial keys that are not functionally dependent can include partial keys pairs that do no fit a functional dependency defined for other tuples in the table. The exception list can represent errors in the one or more data models.

[0011] In determining whether pairs of partial keys are functionally dependent, the method may include defining function F from each partial key to every other partial key to its right in a reordered table for each row in table T, and determining a functional dependency exists for i from1 to (d – 1) and j from (i +1) to d. The function F(Kri) = Krj may be the same function for each tuple t(r) in the table. When for i from 1 to (d – 1) and j from (i +1) to d, the function F(Kri) = Krj is not the same function for each tuple t(r) in the table, and there may exist one or more mappings from Kri to Krj for different values of r. Different values of r can be related to different tuples t(r), and upon determining multiple mappings, the method may include checking whether one or more entries in set {Krj} are similar for each t(r).

[0012] A similarity can be defined with a similarity function and/or a data cleansing function. If a subset of {Krj} is similar, the subset may be compressed to a single value x to compress multiple mappings to a single functional dependency. If a subset of {Krj} is not similar, an exception list for non-similarities may be created. The creation of an exception list for non-

similarities may involve mapping a row number r for tuple t(r) of each dissimilar entry Krj to a corresponding value Kri.

[0013] The method may also include rewriting one or more queries against the table to check the exception list before accessing function F. If no entry exists for the current row in that list, the functional dependency defined by F may be used.

[0014] In another implementation, an article comprising a machine-readable medium storing instructions operable to cause a machine to perform operations that include reducing redundancy within a data model in a database, in which the data model is represented by at least one table. The reduction of redundancy includes determining a number of distinct values of partial keys in a table, and reordering one or more columns of the table by cardinality of partial keys, in which each partial key represents at least one row in the table. The cardinality of a partial key represents a number of distinct values of the partial key. The reduction of redundancy also includes determining whether pairs of partial keys are functionally dependent, and eliminating one or more columns having functional dependencies from the table.

[0015] The systems and techniques described here may provide one or more of the following advantages. For example, one or more methods described can permit a business that stores large amounts of data to reduce an amount of memory required to store the data to realize performance benefits, such as increasing the speed of business processes or reducing network traffic. The methods may permit a business to store the data in less space on databases or other machine-readable media. The amount of resources (e.g., equipment, personnel, money, facilities) for storing and managing data may be reduced, as well as data-related costs. Redundancy reduction may be performed automatically (e.g., without human intervention) or semi-automatically (e.g., with limited human intervention). The systems and techniques may provide insight for one or more users of the data for improving the manner in which the data is organized in data models or

in a data schema (e.g., the structure of a database system, usually with tables, fields in each table, and relationships between the fields and tables).

[0016] The details of one or more implementations are set forth in the accompanying drawings and the description below. Other features and advantages will be apparent from the description and drawings, and from the claims.

DRAWING DESCRIPTIONS

[0017] FIG. 1 illustrates a table in a business database.

[0018] FIG. 2 shows a first method to reduce redundant data.

[0019] FIG. 3 is a flow diagram for part of a second method.

[0020] FIG. 4 is a flow diagram combining the methods of FIGS. 2-3.

[0021] Like reference symbols in the various drawings may indicate like elements.

DETAILED DESCRIPTION

[0022] The present disclosure describes systems, methods, and techniques in which a business can reduce the redundancy of stored data. The data may be stored in tables in a relational schema and may be in memory, a database, or in other machine-readable mediums. The relational schema may include a cube-like data model. In one method, the redundant data may be data that a user can eliminate by means of normalization. In an alternative method, the redundant data may be data for which an approach to normalization fails due to a number of false entries in the data set and/or outliers (e.g., data which is far removed in value from others in a data set). In the second method, similarity-based or data cleansing functionality can be used to generate an exception list that can be of assistance in reducing the amount of redundant data. The data reduction methods, such as a method for reducing data redundancy by reorganizing the

data, may reduce the resources required to handle and store the data, and may reduce the data-related operating and storage costs of a business.

[0023] Fig. 1 shows a schematic example 100 of a table T that may be used in a business database. The table has several rows (170-190) and several columns (110-160). Columns 110-140 may contain key attributes and columns 150-160 may contain non-key attributes. Key attributes (or partial keys) form keys that identify rows, whereas non-key attributes (or key figures) supply information. For example, the partial key K11 in the table (column 110, row 170) can represent a name of an employee (e.g., "Juan Smith"). The partial key K1d (column 130, row 170) can represent a location of the employee (e.g., "Highpoint, NC"). In column 150, row 170, the key figure N11 can represent a record the monetary value of the office supplies consumed by that employee (e.g., "$2,348.93"). In column 160, row 170, the key figure N1k can represent a record a running average of the hours worked by that employee per week (e.g., "35 hours").

[0024] In Fig. 1, an exemplary schematic table T shows n tuples (also called rows) and (d + k) columns. The tuple t(r) on row r 180 of the table includes d key attributes or partial keys Kr1, ..., Krd and k non-key attributes or key figures Nr1, ..., Nrk. The first tuple t(1) is the first row 170 [K11, ..., K1d, N11, ..., N1k]. The n-th tuple t(n) is the last row 190 [Kn1, ..., Knk, Nn1, ..., Nnk]. Values Kri (for i ranging from 1 to d) may be character strings or various kinds of numbers, but in general, they can have a binary representation as bit strings. For the first tuple in the table, the key is the string obtained by concatenating the respective values K11, ..., K1d (where the dots represent the partial keys between the first partial key K11 and the last partial key K1d). If d = 1, the first column contains a unique key attribute and there are no partial keys.

[0025] A difference between a partial key (a key attribute) and a key figure (a non-key attribute) is that a partial key helps to determine the identity of the record specified in its row. If

a key is changed, a different record is specified. In general, a difference between key figures and non-key attributes is that key figures may be aggregated (e.g., to form sums or averages), whereas non-key attributes may not be aggregated in some cases. For example, some non-key attributes may be specified by text strings for which aggregation is undefined.

[0026] The table in Fig. 1 can be normalized or put into a normal form. Three or more kinds of normal form can be recognized. For example, a table is in a first normal form, 1NF, if and only if each cell contains a unique value (which may be empty, or zero, which is also a unique value), and not, for example, a pair or more of values. A table is in a second normal form, 2NF, if and only if it is in 1NF, and any non-key attribute is functionally dependent on the entire key - that is, for any key value, there is exactly one non-key attribute value, and that value depends on each partial key (e.g., that value is not independent of any of the partial keys). A table is in a third normal form, 3NF, if and only if it is in 2NF and there are no functional dependencies between any non-key attributes in the table - that is, if and only if it is in 2NF and the only functional dependencies in the table are from key values to non-key values.

[0027] Fig. 2 shows a method 200 to reduce redundant data in tables. The method 200 eliminates redundancies in tables that are not normalized. One or more terms can be defined for method 200, including the following terms: tuple t, which includes d partial keys K(1), ..., K(d) and k key figures; table T, which includes n tuples (also called rows) and (d + k) columns; function F(x) = y is a mapping between partial keys x and y in the same tuple; and flag fd is a Boolean variable with a value of "true" or "false".

[0028] The method 200 can be executed as illustrated in Fig. 2. For each partial key K(i), determine how many distinct values of K(i) appear in the n rows of table T (block 210). The number of distinct values of K(i) is called its cardinality. Then, reorder the key columns of table

T by cardinality (block 210), so that the column with highest cardinality appears in the leftmost position and the column with lowest cardinality appears in the rightmost position.

[0029] The next step in the method 200 is to check for functional dependencies between each pair of partial keys. For each partial key K(i) in turn (block 220), and each further partial key K(j) to its right (block 225) in the reordered table T, set flag fd to "true" (block 230) and work in turn through all the n tuples t(r), for r running from 1 to n, in the table T (block 240). In order words, loop through successive values of i from 1 to (d-1) (block 220), and loop through successive values of j from (i+1) to d (block 225). Then set the flag fd to "true" for current values of i and j (block 230), and loop through all n tuples t(r) in table T (block 240).

[0030] The next step in method 200 is to define a function F from partial key K(i) to partial key K(j) for the successive rows t of table T. For each row t(r), determine whether the function F has already been defined for the value Kri of partial key F(i) appearing in row r (block 250). If the function has not yet been defined then set the function F(Kri) to Krj, where Krj is the value of partial key K(j) appearing in row r (block 252). In cases where a value Kri of K(i) has already been assigned to different value Krj of K(j) in an earlier row (that is, for a smaller value of r), a function F cannot be defined (block 255). F may not be defined because by definition, F(x) = y is a function only if for any given x, the value of y is unique. In such cases where F cannot be defined, the flag fd is set to "false" (block 257) and the loop through tuples t(r) is broken off (e.g., the end of the t-loop for these values of i and j) (block 265). In cases where a value Kri of K(i) has already been assigned to the same value Krj of K(j) in an earlier row (that is, for a smaller value of r) (block 260), the function F is confirmed and the algorithm continues the loop to the next tuple t(r) (block 240). If the function F is confirmed for all rows t (block 270), the flag fd is still set to "true" and there is a functional dependency between the two columns (block 274). Otherwise, there is not a functional dependency between columns i and j (block 272).

This process repeats for each pair of columns i and j (block 280) until the entire table T is processed (block 290).

[0031] Any column that the method 200 reports as functionally dependent on another column can be eliminated from the table and reconstituted later dynamically, as required, by means of the application of the relevant function F defined in the method 200.

[0032] Fig. 3 is a variation of the flow diagram of Fig. 2. In general, steps 230-265 of Fig. 2 (block 295) can be replaced by steps 310-390 in Fig 3. In Fig. 3, the replacement steps 310-390 are a different treatment of the mappings F. The combination of the steps 210-225, 310-390 (block 300), followed by steps 270-280 can be considered a generalization of the method 200 of Fig. 2, and are shown in Fig. 4. For example, the method 400 in Fig. 4 can be adapted to create a function F plus an exceptions list, and/or the method 400 can be adapted to detect outliers and/or false entries in the data set.

[0033] The method 400 can be executed as follows: for each partial key K(i), determine how many distinct values of K(i) appear in the n rows of table T (block 210). The number of distinct values of K(i) is called its cardinality |K(i)|. Then, reorder the key columns of table T by cardinality, so that the column with highest cardinality appears first (e.g., in the leftmost position), and the column with the lowest cardinality appears last (e.g., in the rightmost position) (block 210). Next, check for functional dependencies between each pair of partial keys K(i) and K(j), for i from 1 to (d – 1) and for j from (i + 1) to d (blocks 220, 225). At this point, the method is the same as method 200. After step 225, the method is the same as method 300 of Fig. 3.

[0034] In method 300 of Fig. 3, the flag fd is set to "true" for current values of i and j (block 310). To find a functional dependency between partial keys K(i) and K(j), a loop is executed through tuples t(r) in table T, for all r from 1 to n (block 320). At each step, a mapping M is

defined from partial key value Kri to partial key value Krj (block 330). If there is no functional dependency between key values K(i) and K(j), some partial key values Kri are repeated for different values of r but map to different values Krj for the respective values of r. In such cases, the different values of Krj are collected in sets y = {Krj} corresponding to the repeated values of Kri (block 330). When all such sets y have been determined by completing the first t-loop (blocks 320 to 335), the first t-loop terminates and a second t-loop begins (block 340).

[0035] In the second t-loop (blocks 340 to 390), the partial key values Krj in respective sets y = {Krj} for successive tuples t(r) are checked (block 345). If the sets y contain more than one value, the method checks whether the values in the set are sufficiently similar to enable them to be compressed into a single value (block 350). If they can be so compressed, the method compresses them to a unique value Krj (block 355) and sets the value of function F(Kri) = Krj (block 370). If they cannot be so compressed, all the incompressible pairs [Kri, Krj] are added to an exceptions list E (block 360). The flag fd for these values of i and j was initialized to "true" at step 310, but fd is set to "false" in case any pairs are added to the exceptions list E (block 360). This process for the second t-loop (blocks 340 to 390) continues until all tuples t(r) have been checked (blocks 380, 390). At step 390, the processing in method 300 is completed for these values of i and j, and the method continues with step 270 in method 400, as shown in Fig. 4.

[0036] Similarity of entries can be defined by using either a standard or a customized similarity measure, such as a similarity function or a data cleansing function. For example, if two keys map to a table of customers, the table may contain two rows for the same customer – one written as "Mueller" and the other as "Müller" – that are either recognized as similar to each other by the fact that every other attribute in the two rows is identical, and the data may be cleansed manually by deleting one of the two rows. The deletion of redundant rows can have two effects. For one, the deletion can directly reduce the redundancy in the data rows. Secondly,

the deletion can indirectly reduce redundancy by enabling redundant columns to be identified and deleted.

[0037] The exception list E can be considered as exceptions to the mapping from the values Kri of partial key K(i) to values Krj of partial key K(j) for cases when the values of K(i) do not map uniquely to values of K(j). The exception list E may be produced once suitable similarity measures and compression procedures (blocks 350, 355) have been implemented. The exceptions include the cases that prevent a function F from being defined from partial key K(i) to partial key K(j). Given the exception list E, a function F for these partial keys can be defined with the proviso that function F is applicable only for those rows r that do not index entries in the exception list E.

[0038] In another implementation, one or more queries and one or more answers to those queries may be associated with the data in the table. An alternative method may check the exception list E to answer a query before accessing function F. The method may be able to answer the query by searching through the exception list E. If no entry exists for the current row in that list, the method can answer the query by using the functional dependency defined by F to find corresponding values in the table.

[0039] The exception list can be particularly useful in cases where the number of exceptions is low. If data cleansing cannot eliminate exceptional tuples, it may be likely that columns i and j cannot be compressed to a single column. Data cleansing can refer the process of examining irregular entries in a table and either correcting or deleting them. For example, if a table column lists the cities of residence of customers of an online clothing store and includes an entry for "New Yonk" then a data cleansing function may either automatically correct the entry to "New York" or flag the entry for manual examination and correction.

[0040] In general, the methods 200 and 400 can both be executed recursively on dependent tables. Any columns that are reported as dependent can either be eliminated automatically or displayed to the user for interactive individual treatment. Once the first and second methods have been used to eliminate redundancies from a table and the data is manually cleansed to remove any false entries reported as exceptions by the second method, the table can be stored in less space in memory or in the database. The methods 200 and 400 are applicable to any data stored in tables in a relational database model such as a cube-like data model.

[0041] As used herein, the terms "electronic document" and "document" mean a set of electronic data, including both electronic data stored in a file and electronic data received over a network. An electronic document does not necessarily, but may, correspond to a file. A document may be stored in a portion of a file that holds other documents, in a single file dedicated to the document in question, or in a set of coordinated files. The term "object" may refer to information sources such as documents, reports, presentations, files and directories.

[0042] Various implementations of the systems and techniques described here can be realized in digital electronic circuitry, integrated circuitry, application specific integrated circuits (ASICs), computer hardware, firmware, software, and/or combinations thereof. These various implementations can include one or more computer programs that are executable and/or interpretable on a programmable system including at least one programmable processor, which may be special or general purpose, coupled to receive data and instructions from, and to transmit data and instructions to, a storage system, at least one input device, and at least one output device.

[0043] The software (also known as programs, software tools or code) may include machine instructions for a programmable processor, and can be implemented in a high-level procedural and/or object-oriented programming language, and/or in assembly/machine language. As used

herein, the term "machine-readable medium" refers to any computer program product, apparatus and/or device (e.g., magnetic discs, optical disks, random access memory (RAM), programmable logic devices (PLDs)) used to provide machine instructions and/or data to a programmable processor, including a machine-readable medium that receives machine instructions as a machine-readable signal. The term "machine-readable signal" refers to any signal used to provide machine instructions and/or data to a programmable processor.

[0044] The systems and techniques described here can be implemented in a computing system that includes a back end component (e.g., as a data server), or that includes a middleware component (e.g., an application server), or that includes a front end component (e.g., a client computer having a graphical user interface, portal, or a Web browser through which a user can interact with an implementation of the systems and techniques described here), or any combination of such back end, middleware, or front end components. The components of the system can be interconnected by any form or medium of digital data communication (e.g., a communication network). Examples of communication networks include a local area network (LAN), a wide area network (WAN), a wireless local area network (WLAN), a personal area network (PAN), a mobile communication network using a multiple access technology (e.g., a cellular phone network with code division multiple access, CDMA), and the Internet.

[0045] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

[0046] Although only a few implementations have been described in detail above, other modifications are possible. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the claims below. For example, the logic

flow depicted in Figs. 2-4 do not require the particular order shown, or sequential order, to achieve desirable results. Accordingly, other implementations are within the scope of the following claims.